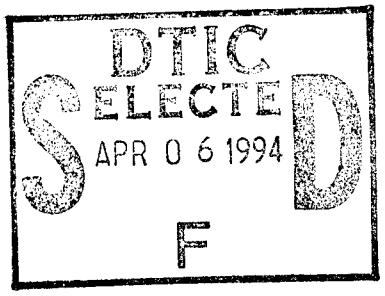
Analysis of Symbolic Parameter Models: A final report

Herbert A. Simon, Thad Polk
& Kurt VanLehn



REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	3-16-95	Final Technical Report, 2/15/91-2/14/93

4. TITLE AND SUBTITLE
Analysis of Symbolic Parameter Models: A final report

5. FUNDING NUMBERS
N00014-91-J-1529

6. AUTHOR(S)
Herbert A. Simon, Thad Polk, & Kurt VanLehn

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Learning Research and Development Center
3939 O'Hara St.
University of Pittsburgh, Pittsburgh, PA 15260

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Department of the Navy
Office of the Chief of Naval Research
800 N. Quincy St., Code 1511:LG
Arlington, VA 22217-5000

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Appendix
To appear in S. Chipman & P. Nichols (Eds.), Alternative Approaches to Diagnostic Assessment. Hillsdale, NJ: Erlbaum.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Distribution unlimited; approved for public use.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Model-fitting, the problem of finding parameter settings that cause a model to fit given data as closely as possible, is a hard but important problem in cognitive science in general, and in cognitive diagnosis in particular. Efficient solutions have been found for certain types of model-fitting problems (e.g., linear and integer programming) that involve specific types of parameters (usually continuous) and models (usually linear). But these techniques usually do not apply to computational cognitive models whose parameters are often discrete and symbolic and whose internal workings must be treated as a black box for the purposes of fitting. We have constructed ASPM (Analysis of Symbolic Parameter Models), a suite of computational tools for fitting and analyzing such symbolic parameter models. We have shown how it can be used to fit and analyze computational models with well over 10 billion combinations of parameter settings. We have made a good start towards making ASPM robust and user friendly in preparation for releasing it for public use.

14. SUBJECT TERMS
model-fitting, analysis of symbolic parameter models, ASPM, symbolic parameter models

15. NUMBER OF PAGES
32

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
unclassified	unclassified	unclassified	UL

Analysis of Symbolic Parameter Models: A final report

Herbert A. Simon
Department of Psychology
Carnegie Mellon University

Thad Polk
Department of Psychology
University of Pennsylvania

Kurt VanLehn
Learning Research and Development Center
University of Pittsburgh

March 29, 1995

Abstract

Model-fitting, the problem of finding parameter settings that cause a model to fit given data as closely as possible, is a hard but important problem in cognitive science in general, and in cognitive diagnosis in particular. Efficient solutions have been found for certain types of model-fitting problems (e.g., linear and integer programming) that involve specific types of parameters (usually continuous) and models (usually linear). But these techniques usually do not apply to computational cognitive models whose parameters are often discrete and symbolic and whose internal workings must be treated as a black box for the purposes of fitting. We have constructed ASPM (Analysis of Symbolic Parameter Models), a suite of computational tools for fitting and analyzing such symbolic parameter models. We have shown how it can be used to fit and analyze computational models with well over 10 billion combinations of parameter settings. We have made a good start towards making ASPM robust and user friendly in preparation for releasing it for public use.

1 Objectives

In 1991, ONR awarded grants to Carnegie-Mellon University (N00014-91-J-1527) and and the University of Pittsburgh (N00014-91-J-1529) to develop ASPM1. The work continued under a renewal of the CMU grant in 1993. The renewal contract funded the development of ASPM2, which is a successor to ASPM1 designed for use by the cognitive science community. The attached report (Newell et al., 1995), which is a pre-print of a chapter to be published in a book edited by Susan Chipman, describes the two systems and their particular approach to the parameter fitting problem. It also introduces terminology that is used in this final report.

There were several objectives to the effort:

- ASPM1 was designed for one important task: calculating the best-fitting parameters for a model given some data. However, early experience with it showed that developing a model required much more than just calculating best-fitting parameters (see the attached report for examples). Thus, the first objective was to increase the types of calculations that ASPM could perform.

- ASPM1 had only a limited capability to display its analyses. However, displaying data appropriately is very important, especially during exploratory data analysis, as it can greatly facilitate the development of the investigator's understanding. The second objective was to increase the types of displays that ASPM could provide.

- ASPM1 was used only with one model and data set—Thad Polk's model of deductive reasoning. In order to remove idiosyncrasies caused by such a parochial development, it is important to use ASPM with at least one other model and dataset. The classic Debuggy model (Burton, 1982) and Southbay data set (VanLehn, 1990) were chosen, in part because this would allow a comparison between ASPM's exact calculations and Debuggy's more heuristic calculations.

- ASPM1 had no user manual and only limited help facilities. Since we intend to provide only very limited user support once ASPM is released, we must provide some external and internal documentation of the system.

Each of these objectives is discussed in turn.

2 Improving the functionality of ASPM

The main impediment to increasing the functionality of ASPM1 was that it used many different data structures, all of which represented partitions of the parameter space. The original idea behind ASPM2 was to use just one data structure, and allow the users to apply any operation to it. This would simplify the users' task considerably and yet give them even more power than before by allowing them to compose operations in arbitrary orders.

As work began, we discovered that our original plan went too far. Although ASPM1 was too structured, the envisioned version of ASPM2 was too unstructured. Specifically, the ideal of freely applying operations was not realizable in practice. For example, it does not make sense to compare a partition of the parameter space to a subject's response unless each region of the partition has a response associated with it. Similarly, one cannot search for a best fit unless the partition has already been compared to a subject's response and sorted. In short, ASPM2 would have many of the same restrictions as ASPM1 in terms of what operations can be applied when. But these conditions would be invisible to the user and difficult to detect by ASPM2, making it harder to use and build.

Based on our previous experiences, we decided that ASPM1's data structures should be replaced by the following three:

- Response partitions. These are partitions of all or part of the parameter space, where each region of the partition is associated with a unique response. That is, all parameter settings that produce the same response from the model are placed in the same response region. Unlike ASPM1, there is no distinction between primitive and compound response partitions.

- Fit partitions. These are partitions of all or part of the parameter space, where each region of the partition is associated with a unique output from the user's fit measure. That is, different parameter settings are in the same fit region even when they produce different model responses, as long as those model responses fit the subject's response equally well.

- Generic regions. These are arbitrary subsets of parameter space with no associated information. These are often used to restrict ASPM's attention to a subset of the parameter space, although they can be used for many other purposes as well.

As in ASPM1, considerable care was taken in designing these data structures. Bit vectors were used whenever possible to make the data structures as compact as possible. Redundancy was provided in order to speed up some processes.

Having decided on the data structures, we turned to overhauling the major algorithms. We were particularly concerned with the compose operation, as this is the most computationally expensive part of the system. Given the geometric nature of compose (overlaying or intersecting two partitions), we turned to the field of computational geometry for possible algorithms. The divide and conquer algorithm that we decided upon is described in our first quarterly progress report.

Work has proceeded smoothly since then. However, because we have had to revise almost every module in the whole system in order to propagate the changes in the major data structures, the code is still not quite finished. Dirk Kalp, the main implementer, is putting the finishing touches on the main operations (compose, sort and search) by working evenings and weekends without pay.

3 Display functions

Data analysis, especially exploratory data analysis, is quite difficult without good ways of displaying the results of the analysis. For instance, a scattergram tells one different things about a regression than the beta coefficients and the residuals. Part of the job of making ASPM useful involved giving it better ways of displaying the data.

Given the discrete nature of ASPM and the lack of a uniform graphics language for workstations, we decided to use character-oriented graphics rather than pixel-oriented. Thus, all of ASPM's displays can be displayed by anything that can display Ascii text, and they can be printed to files or shipped easily across the network. Although this may seem old-fashioned, we did not want ASPM to become obsolete because it used a type of graphics that was no longer supported. We believe that character-oriented graphics will remain alive for the foreseeable future.

The displays supported by ASPM2 are the following:

```
resp-part ..... displays a response partition.
resp-parts ..... displays a list of response partitions in the expt.
fit-part ..... displays a fit partition.
fit-parts ..... displays a list of fit partitions in the expt.
gen-region* ..... displays a generic region.
gen-regions* ..... displays a list of generic regions in the expt.
params ..... displays the current parameters for the expt.
pspace ..... displays the parameter space.
param-names ..... displays the names of selected parameters.
pval-names ..... displays the names of parameter values.
tasks ..... displays the current set of tasks for the expt.
subjs ..... displays the current set of subjects for the expt.
responses ..... displays subject responses to the current tasks.
coupling ..... displays the task/parameter coupling matrix.
```

The display commands with asterisks are still not quite ready. Thad Polk is finishing up their implementation.

4 Applying ASPM to Debuggy

It became clear early on that ASPM2 would not be ready fast enough for the Debuggy analysis to be done using it, so a preliminary analysis was done with ASPM1. The experience gained from this has been put to good use in reconceptualizing the design of ASPM.

The basic job of Debuggy is to find a best-fitting set of bugs to match a given student's responses to a subtraction test. (This task is described in the introduction of the attached report.) To convert this model to the ASPM framework, we first had to redesign it so that it had parameters with discrete values. A simple way to do this would be to have a single parameter for each bug: the parameter is 1 if the bug is present and 0 if the bug is not. Given that there are 127 bugs in Debuggy, this would yield a parameter space of $2^{127} = 1.7X10^{38}$, which is large even by ASPM standards.

However, this parameterization didn't make much sense, because certain combinations of bugs simply cannot occur together—they are logically incompatible. For instance, the bugs $0-N = N$ and $0-N = 0$ cannot both apply to the same model, because they apply to the same columns but write different answers in those columns. Therefore, we designed a set of parameters such that the values of those parameters were mutually exclusive bugs. That is, $0 - N = N$ and $0 - N = 0$ became different parameter values for the same parameter. This considerably reduced the size of the parameter space.

The major task for this project was to build a piece of software, called Deb, that would output a response partition for a single subtraction problem. That is, it would calculate the answers to that problem given all possible combinations of bugs. Thus, it actually had to visit each point in the parameter space, run the model, and produce the model's response for that particular combination of bugs.

Our first approach was a brute force one. We implemented a depth-first generator, based on the Prolog language because it has built-in capabilities for depth-first search. By the time we had equipped Deb with the 27 most common bugs, its brute force search was running too slowly.

We tried a variety of search techniques in order to speed up the generation. The basic idea was to try to remove redundancies in the generation. For instance, suppose we have only two parameters, A and B. During brute force search, we first set A to 1 then generate model responses for each value of B. We next set A to 2 and generate model responses of each value of B. But suppose that this second set of responses is the same as the first set. If we had known that A and B were independent, then we could have saved the time used to generate the second set of responses. But how could we know when A and B would be independent in this fashion?

Working with Anandeep Pannu, a University of Pittsburgh graduate student, we finally discovered a technique that could detect independencies such as the one illustrated above, and avoid redundant generation. The Deb program was rebuilt to include this "smart" generator.

Unfortunately, the definition of "independent" turned out to be so narrow that redundancies were seldom found. The "smart" Deb took considerably longer than the brute force Deb, because it had to maintain some complicated data structures in order to detect redundancies. In our post-hoc analysis, it did not seem that savings from detecting redundancies

would outweigh the extra overhead even when the parameter space was expanded. Thus, we abandoned this approach for Deb. However, it remains a viable approach for task domains which have more redundancy. The "smart" generation algorithm will be described in a forthcoming journal article on ASPM.

On examining the independencies that Deb found, we discovered that some of them reflected underlying "permanent" independencies in the parameter space that should have been apparent initially. We redesigned the parameterization in order to take advantage of these independencies. This considerably reduced the size of the parameter space, and allowed the brute force search to generate responses for the whole parameter space on even the most complex subtraction problems in the data.

With the response partitions finally generated, we began to analyze them with ASPM1. We soon discovered that the response partitions were full of mistakes. The same parameter settings occasionally occurred in multiple response regions, and some parameter settings appeared in no response regions. We realized that other users would also have trouble generating legal data for ASPM. In response, a function for checking data integrity was added to ASPM.

With this added functionality, the bugs in Deb were located and fixed, and the Debuggy data were analyzed. The results were presented at an ONR contractors conference, and summarized in our second quarterly progress report.

Work on Deb stopped until recently, when ASPM2 became operational. Fortunately, the Deb program itself aged well: it still runs. However, its output needs to be reformatted for the new ASPM data structures. Kurt VanLehn is finishing up this part of the work.

5 User manual, Journal articles, etc.

The user manual for ASPM2 has not yet been written, but a concise help command has been added to the system, and the internal documentation of the program has been extensively revised. In particular, the data structure format is extensively documented, including examples. This should make writing the user manual fairly straightforward.

A journal article summarizing all the ASPM research is in progress, based on a rough draft written some years ago. It will be extensively revised in order to bring it up to date with respect to ASPM2, and to incorporate the results of the Deb experiment.

We anticipate considerable interest in ASPM in the near future, as Thad Polk's deductive reasoning work has just been accepted for publication in *Psychological Review*, arguably the most prestigious journal in psychology (Polk and Newell, 1995). Given that the article presents the results of a number of ASPM analyses (that would have been impossible without the system), we expect to get some "customers" when that article appears.

6 Summary

On the whole, we have gotten about 90% of the work done that we intended to do in order to get ASPM ready for external distribution. There are still some bits of programming left to do, the user manual must be written, and the journal article must be completed. However, the rest of the system seems to be functioning as planned.

7 Papers supported by the grant

This grant supported the writing of a journal article (Polk and Newell, 1995), a book chapter (Newell et al., 1995), and an unpublished master's thesis (Pannu, 1993). Two further publications are in preparation: A journal article presenting all the ASPM work and a user manual.

References

Burton, R. (1982). Diagnosing bugs in a simple procedural skill. In Sleeman, D. and Brown, J., editors, *Intelligent Tutoring Systems*. Academic Press, London.

Newell, A., Polk, T., and VanLehn, K. (1995). Analysis of symbolic parameter models. In Nichols, P., Chipman, S., and Brennan, S., editors, *Cognitively diagnostic assessment.* Lawrence Erlbaum Associates, Hillsdale, NJ. in press.

Pannu, A. S. (1993). Search strategy for exhaustive generation of parameter settings for a student model. Master's Thesis, Intelligent Systems Program, University of Pittsburgh, Pittsburgh, PA.

Polk, T. and Newell, A. (1995). Deduction as verbal reasoning. *Psychological Review*, 102(3). in press.

VanLehn, K. (1990). *Mind Bugs: The origins of procedural misconceptions.* MIT Press, Cambridge, MA.

(in press) S. Chipman & P. Nichols (Eds), Alternative Approaches to Diagnostic Assessment. Hillsdale, NJ: Erlbaum.

ASPM2: Progress Toward the Analysis of Symbolic Parameter Models

Thad A. Polk
Department of Psychology
University of Pennsylvania

Kurt VanLehn
L.R.D.C.
University of Pittsburgh

Dirk Kalp
School of Computer Science
Carnegie Mellon University

Running head: ANALYSIS OF SYMBOLIC PARAMETER MODELS

Abstract

Model-fitting, the problem of finding parameter settings that cause a model to fit given data as closely as possible, is a hard but important problem in cognitive science in general, and in cognitive diagnosis in particular. Efficient solutions have been found for certain types of model-fitting problems (e.g., linear & integer programming) that involve specific types of parameters (usually continuous) and models (usually linear). But these techniques usually do not apply to computational cognitive models whose parameters are often discrete and symbolic and whose internal workings must be treated as a black box for the purposes of fitting. We present ASPM (Analysis of Symbolic Parameter Models), a suite of computational tools for fitting and analyzing such symbolic parameter models, show how it can be used to fit and analyze computational models with well over 10 billion parameter settings, and describe a few changes in the initial design that will make it even more powerful as well as easier to use.

ASPM2: Progress Toward the Analysis of Symbolic Parameter Models

Suppose you wanted to improve how subtraction is taught. Clearly, in designing an appropriate curriculum, you may want to understand how children go about solving the task and why they make the mistakes they do. One approach to developing such an understanding would be to build, test, and refine computational models of subtraction behavior. Of course, different children approach subtraction in different ways, so your model would need to include parameters that allow its behavior to be tailored to that of individual students. For example, some students fail to borrow and always subtract the smaller digit in a column from the larger even when the smaller digit appears in the upper number:

```
  42
- 18
  --
  36
```

Thus, you would probably want to include a parameter that controls whether your model makes this error:

```
Parameter: Smaller-from-larger
Values:    (a) makes this error
           (b) does not make this error
```

Similarly, many children treat blanks as if they were 1's and make errors like the following:

```
  35
-  2
  --
  23
```

So you might include another parameter, Sub-one-over-blank, that controls

whether or not your model behaves this way. Some mistakes may be due to more than one error. For example, the following behavior could be explained by assuming that the student made both errors described above:

```
  671
-  28
  ---
  557
```

In addition to these two, Brown & Burton (1978) identified a large number of other common subtraction errors, which they called "bugs" (also see Burton, 1982; VanLehn, 1990). For a subtraction model to predict the behavior of different students with any accuracy, it would seem to require a fairly large number of parameters.

But this raises an entirely new set of issues. Which parameters are important in predicting behavior and which are not? How much would it affect the model's fit if a parameter were removed or if a value was changed? How "good" is the model's fit? How does one avoid overfitting the data? And perhaps most importantly, which parameter settings best fit a given student's behavior and how can you figure this out without exhaustively trying the thousands, millions, or even billions of possible combinations of parameter values?

The same issues must be faced by computational models in a wide variety of domains: subtraction, physics, medical diagnosis, deductive reasoning, design, etc. Indeed, analyzing any model of behavior in which there are a large number of variables that affect performance will require addressing these problems.

Powerful techniques have already been developed for fitting and analyzing certain types of parameterized models (e.g., linear and integer programming). These techniques impose certain constraints on the model being analyzed (e.g.,

that it is linear) and on its parameters (e.g., that they are continuous or at least ordered) so that efficient algorithms can apply. Unfortunately, the types of computational models mentioned above do not usually satisfy these constraints. Parameter values are usually symbolic and unordered and the behavior of the model is not simply a linear combination of parameter values, but is controlled by a complicated computer program. No efficient techniques currently exist for fitting and analyzing such models. ASPM1 and its successor ASPM2 (Analysis of Symbolic Parameter Models) are sets of computational tools that provide such a capability for certain common types of symbolic parameter models.

Fitting and Analyzing Symbolic Parameter Models

It is possible to characterize parameterized models in fairly abstract terms. For a given task, the model maps each parameter setting (a point in parameter space) onto a predicted response. For example, if the task were a specific subtraction problem, then a parameter setting might correspond to a set of subtraction bugs from which the computational model predicts a specific response to the task. Different parameter settings that predict the same response for the task can be grouped together to form response regions within parameter space (e.g., the same response to a subtraction problem could arise from a variety of different bugs). The set of all such response regions forms a response partition of parameter space[1]. Figure 1 shows the situation when the parameter space is 2-dimensional (i.e., when it contains only two parameters). The parameter space (the square on the right) is divided into disjoint response regions thus forming a response partition. The model maps all parameter settings within a given response region onto the same predicted response. For example, all settings in region RR1 map onto response r1 in the figure. A given response may have an empty response region (e.g., r4 and r5) in which case the model is unable to produce that response on that task — the response is outside the model.

In general, one wants to analyze behavior on a set of tasks, rather than on a

---

Insert Figure 1 about here

---

single, isolated task. If a student correctly solves a single subtraction problem, for example, that does not imply that the student has mastered subtraction. The student could have bugs (e.g., in borrowing) that fail to show up on a given problem (e.g., that does not require borrowing). And given the wide variety of common subtraction bugs, it is impossible for behavior on any single task to diagnose the presence or absence of each of them. More generally, the responses of a subject to a wide range of tasks are much more diagnostic about what underlies that subject's behavior than is the response to a single task. Thus, the above analysis needs to be generalized to deal with multiple tasks.

One way to achieve this goal is by treating multiple tasks as single compound tasks which are made up of a sequence of primitive tasks (Figure 2). Parameter space is once again partitioned into response regions. But instead of mapping onto the same primitive response, each response region now maps onto the same compound response, that is, a specific sequence of primitive responses for each of the primitive tasks that together comprise the compound task. Similarly, the response regions and response partition could be called the compound response regions and compound response partition respectively. In general, we will simply use the terms task, response, response region, and response partition except when we want to draw attention to the primitive/compound distinction.

Insert Figure 2 about here

Given a response partition for a task (either primitive or compound), the

analysis of a model's fit to a subject's behavior becomes straightforward. If one of the model's predicted responses matches the subject's response exactly, then the associated response region contains all the best-fit parameter settings for that subject on that task (and only those settings). In fact, they are exact-fit settings. More generally, one can allow for inexact matches by defining a fit measure between subject responses and model predictions. Perhaps the simplest such measure is the Hamming distance — the number of tasks on which the model incorrectly predicts the subject's response. But in many situations more sophisticated fit measures would be desirable. For instance, if certain tasks provide more reliable information for diagnosis than others, then it would be appropriate for the fit measure to weight the model's success on those tasks more heavily. In other cases, one might want the fit measure to correspond to a probability (e.g., how likely is the subject's observed response based on the model's predicted response and some error model).

Using such a fit measure to compare the subject's response with each possible model prediction, one can find the response region (or regions) whose associated prediction most closely resembles the subject's behavior (according to that measure). The response regions can even be sorted based on the fit measure. Response regions whose associated predictions fit the subject's behavior equally well are merged to form fit regions and together these fit regions form a fit partition of parameter space (Figure 3). The best-fit settings will all be at one end of this partition while the worst-fit settings will be at the other. Such a fit partition provides the basis for answering a variety of questions about the model and subject: the quality of the model's fit, the stability of that fit with specific parameter changes, the scope of the model's predictions (e.g., whether it can exactly match a given behavior), and so on. Indeed, if one could compute a fit partition for the compound task consisting of all primitive tasks in a domain, one would have virtually all relevant information for analyzing the relationship

between a subject's behavior and a model's predictions (at least with respect to the fit measure that was used).

Insert Figure 3 about here

For example, such fit partitions provide a means for comparing different models and determining just how good a model really is. Before such an analysis is possible, however, it is necessary to control for the number of available degrees of freedom. If a model has too many degrees of freedom then it may be able to fit any data (including random noise) and it may overfit real data. One way of dealing with this problem is to assess the model's fit to data that is separate from the data used to set the parameters. For example, if one could compute a model's fit partition with respect to a subject's behavior on half the tasks, then the resulting best-fit parameter settings could be assessed against the subject's behavior over the remaining tasks. Such 0-parameter fits can be used to compare different models with varying degrees of freedom in order to determine which model has more predictive value. Similarly, 0-parameter fits can be used to compare a model with various reference theories including a random model (to ensure that the model does better than chance), a model corresponding to the modal responses of subjects (to determine whether the model is capturing individual differences over and above the common responses), and a test-retest model in which each subject's behavior at a different time serves as the model (to estimate how close the model is to optimality). See Polk (1992) and Polk & Newell (submitted) for examples of these types of analysis.

ASPM1

ASPM (Analysis of Symbolic Parameter Models) is a set of computational tools for computing response partitions and fit partitions. The original version

(ASPM0) was built in Lisp and was strongly tied to a specific model (of syllogistic reasoning, Polk, 1992; Polk & Newell, submitted). It was only after building ASPM0 that we realized the general utility of the ideas. ASPM1 was our first attempt to construct a general-purpose version of the tools.

ASPM1 was built in C. It had three major operations which correspond closely to the analysis presented above:

1. Compose took response partitions for a sequence of primitive tasks as input and produced as output the response partition for the compound task corresponding to that sequence of primitive tasks.

2. Sort took as input a compound response partition and a subject's behavior on that compound task and produced as output a fit partition. It used a specific, built-in fit measure (Hamming distance) to compare subject responses with model predictions.

3. Search took fit partitions from different compound tasks as input and produced as output all parameter settings that best-fit the subject on the compound task corresponding to the sequence of input compound tasks (i.e., the best-fit region). This operation was used to find best-fit settings for compound tasks that were too large for a complete fit partition to be computed in a reasonable amount of time.

In order to make use of ASPM1, we first had to compute primitive response partitions which could serve as inputs (they could then be composed together, sorted, and searched). ASPM cannot compute these on its own, since they store information that is specific to the model being analyzed (its predictions for each

task given any parameter setting), so it is up to the user to provide them. In the worst case, computing the response partition for a primitive task could involve running the model using every possible parameter setting and storing the results. Such an approach is computationally infeasible for models with a large number of parameter settings. Fortunately, for many models such an exhaustive approach is unnecessary. In many cases, only a subset of parameters is relevant to an individual task, a characteristic we refer to as loosely-coupled tasks. In subtraction, for example, parameters that control borrowing behavior are irrelevant to subtraction problems that do not involve borrowing. In domains with loosely-coupled tasks, it becomes feasible to compute the primitive response partitions since irrelevant parameters can be ignored. One need only run the model using all the possible parameter settings involving the relevant parameters — the values of the other parameters are irrelevant and need not be varied.

Using ASPM1 we were able to compute best-fit settings from parameter spaces of over ten billion total settings and to perform numerous other analyses of interest. For example, in the domain of syllogistic reasoning, we were able to (1) compute a model's complete set of best-fitting parameter settings for 103 subjects, (2) compute 0-parameter fits for all these subjects by using a subset of tasks to set the parameters and the remaining tasks to assess the quality of the fit, (3) identify a few parameters that were most important in achieving a high degree of fit and use these to produce a simpler, but still accurate, model, (4) analyze the range of the model by fitting it to artificial data (e.g., random data to ensure that the model could not fit everything, perfect performance to determine if the model allowed for rationality, and (5) determine the predictive value of specific theoretical assumptions by comparing fits from parameter settings that either did or did not incorporate those assumptions (see Polk, 1992; Polk & Newell, submitted).

Despite its power, in using ASPM1 we came up with a number of ideas for

its improvement:

1. The distinction between primitive and compound response partitions should be transparent to the user. ASPM1 represented and processed primitive and compound response partitions differently. As a result, operations that processed one type of response partition were unable to process the other. For example, compose only worked on primitive tasks, not compound tasks. Consequently, the user was unable to use the output of one compose operation as the input to another. As discussed above, there is no reason why primitive and compound response partitions cannot be treated identically and this was an unnecessary limitation.

2. The system should be able to process arbitrary subsets of parameter space in a straightforward manner. ASPM1 dealt primarily with the entire parameter space — both response and fit partitions divided all of parameter space into individual regions. But there are a number of questions that are best answered by restricting attention to a subset of parameter space. For example, how would removing a few parameters or parameter values affect the fit (requires restricting parameter space by excluding those values)? What is the range of predictions produced by the best-fit settings (i.e., what does the response partition look like when restricted to include only best-fit settings)? What predictions does a particular setting produce for a specific primitive task or for a set of tasks (requires restricting a primitive or compound response partition to include only that

setting)? ASPM1 was eventually augmented with the ability to restrict primitive response partitions based on a subset of parameter space, but a more general facility should really be provided.

3. Best-fit regions should be treated as fit partitions. The final result of many analyses in ASPM1 was a best-fit region — a distinct data structure representing all the best-fitting parameter settings. But best-fit regions are just a special case of fit partitions; they are fit partitions that are restricted to best-fit settings and that consequently have only a single fit region. ASPM should treat them as such since that will allow any operations that process fit partitions to process best-fit regions as well (e.g., the search operation and functions that manipulate subsets of parameter space).

ASPM2

Based on these ideas, we have designed and are now implementing ASPM2 — a new version of the system. In place of ASPM1's four central data structures (primitive response partitions, compound response partitions, fit partitions, and best-fit regions), ASPM2 will have only two: response partitions (for both primitive and compound tasks) and fit partitions (of which best-fit regions will be a special case). In addition, ASPM2 will have one additional data structure, generic regions, for representing arbitrary subsets of parameter space. This additional data structure will allow users to create parameter subspaces from scratch or based on existing response or fit regions, to process these regions using basic set operations (union, intersection, set-difference, etc.) and to compute new restricted response and fit partitions that only contain settings from the specified generic region.

In order to make these ideas more concrete and to illustrate their power, we will now consider a few specific analyses that were either difficult or impossible to perform using ASPM1, but that will be simple using ASPM2.

Example #1: Recovering predictions from specific settings

Suppose one wanted to know whether all of the best-fit settings for a subject produce identical predictions or if they should really be broken up into separate groups, each of which fits the subject equally well, but for qualitatively different reasons. This kind of analysis is critical in analyzing and improving a computational model. For example, if all the best-fit settings are failing to successfully predict the same tasks, then that suggests that trying to improving the model's accuracy on those tasks could greatly improve the model. On the other hand, if different best-fits make inaccurate predictions on completely different tasks, then focusing effort on any one of them would not have as big a big payoff.

What is needed is to be able to recover the predictions for each of the best-fit settings (hopefully, without having to run the model on all of them and record the results). This information is exactly what would be provided by a response partition that had been restricted to include only best-fit settings. Different regions within that partition would correspond to qualitatively different ways of achieving the same level of fit while settings within a region would be known to produce identical predictions. Thus, the number of regions within such a response partition would correspond to the number of distinct best-fitting predictions that could be produced by the model. Computing such a best-fit response partition will be straightforward in ASPM2:

1. Compute the best-fit settings. Use compose, sort, and, if necessary, search to produce a fit partition containing the best-fit region.

2. Create a generic region corresponding to the best-fit region.

3. Restrict the response partitions produced by compose in step 1 above so that they only contain settings from the best-fit generic region.

4. If there is more than one such response partition, compose the resulting restricted response partitions together to form a single best-fit response partition.

Such an analysis would have been impossible in the original version of ASPM1 since it provided no facilities for creating generic regions or restricting response partitions (steps 2 and 3 above). Because the need for this type of analysis arose so often (e.g., in order to determine what parts of the model needed the most work; see the discussion above), ASPM1 was eventually augmented with the ability to restrict primitive response partitions (but not compound) on the basis of best-fit regions (but not other regions). But this functionality was severely limited; it could only use best-fit regions, not second best-fit or worst-fit regions, and it could only restrict primitive response partitions not compound response partitions. In contrast, all these operations would be simple using ASPM2.

Example #2: Imposing constraints on ASPM results

In many situations, one wants to impose additional constraints on the results produced by ASPM. For example, one might want to know how the other parameters should be set if parameter 3 is fixed at value "c" and parameter 7 is fixed at value "a" (instead of letting all the parameters vary). Or one might want to compute the best-fit settings that are guaranteed to correctly predict behavior on a subset of tasks, instead of just the generic best-fit settings. These and similar situations will be handled easily in ASPM2 by restricting the parameter space before computing best-fit settings. In the case of fixed parameter values, this will

be particularly straightforward: create a generic region containing the entire parameter space except with the appropriate parameters fixed and then restrict the response and/or fit partitions that have already been computed. Computing best-fit settings that are guaranteed to correctly predict behavior on a subset of tasks will be only slightly more complicated:

1. Use compose to compute the compound response partition for the subset of tasks that the model must predict correctly.

2. Sort the compound response partition to identify the parameter settings that correctly predict behavior on these tasks.

3. Create a generic region corresponding to these settings.

4. Restrict the response and/or fit partitions that have already been computed for the larger space based on this generic region.

These analyses are not possible in ASPM1 because of its limitations in creating and manipulating generic regions and because it cannot restrict compound response partitions and fit partitions based on generic regions.

Future Work

ASPM2 is currently being implemented and promises to offer a powerful set of tools for analyzing symbolic parameter models. But if ASPM2 is going to come into routine use by a variety of researchers as we hope, it will have to be extremely easy, even trivial, to use. We have identified three major ways to improve ASPM's usability:

1. The system should aid the user in computing primitive response partitions. The major user input to ASPM1 was a set of primitive response partitions and the system did not provide any help in their construction. ASPM would be easier to use and analyses could be

begun more quickly if it provided some tools for aiding the construction of primitive response partitions.

For example, our experience is that debugging and improving a model requires running its output through ASPM many times. In the syllogistic reasoning work, for instance, the results of an ASPM analysis would lead to insights into how to improve the model (e.g., at one point, we found that the model was particularly bad at predicting behavior on tasks involving the quantifier "All". We subsequently modified the model to include an new strategy on these tasks and the fits improved dramatically). But as soon as the model changed, the primitive response partitions would no longer be valid and the model would need to be run again. Under such conditions, it is very easy to mistakenly intermingle data analyses from different versions of the model. ASPM should keep track of all runs of the model in order to, for instance, prevent the user from accidentally composing primitive response regions created by different versions of the model. This may require that the model be called from inside ASPM.

Another problem is that generating all possible responses to a task can be combinatorially complex. ASPM cannot help with this directly, as such generation is the job of the user's model, but we have discovered some approaches that significantly reduce the combinatorics in our test task domains (VanLehn & Pannu, in preparation). The documentation of ASPM should describe such

techniques, and provisions should be made to disseminate other such techniques as they are developed.

2. The system should allow for the use of different, user-specified fit measures. ASPM1 had a fixed fit measure built in: the number of matches between the model and subject on the primitive tasks (Hamming distance). But, as discussed earlier, different types of analyses require different fit measures and ASPM should allow for this diversity (see

3. The system's command syntax should facilitate the construction of complex scripts and even full-fledged computer programs for controlling sophisticated analyses. ASPM should make it easy for the user to use command scripts that include iteration, conditionals, and other programming language constructs. The UNIX shell in which ASPM1 ran provides these capabilities, but ASPM1 made them difficult to exploit since the commands were designed to be interactive. ASPM should provide a simple command syntax including command line arguments that can be used when the user does not want to interact with commands.

Conclusion

We are confident that ASPM2 will provide a very powerful set of tools — a sort of scientific workbench — for fitting and analyzing certain types of parameterized models. Of course, it is not a panacea and will be more helpful in some domains than in others. In trying to understand ASPM's scope of applicability, we have identified the following limitations to its use.

1. Discrete symbolic parameters: The model must have a finite set of parameters, with each parameter's values being a separate discrete set of symbols. This restriction is necessary so that response and fit partitions will not divide up parameter space into an infinite set of regions. In particular, ASPM cannot deal with continuous parameters (such as real numbers).

2. Constant subject parameters: The model must have parameters whose values can be assigned to describe a subject. The parameter set must apply to all subjects on all tasks the subjects perform. Each subject should be described by the same set of parameter values for all tasks the subject performs. Without this restriction ASPM would need to consider all possible combinations of parameter settings from task to task which is not computationally feasible. Thus, if a subject's behavior could best be described as switching between parameter settings, the parameter space would need to be changed to make this variability explicit.

3. Loosely-coupled tasks: Only a small subset of parameters can be relevant to each of the primitive tasks the subjects perform. This restriction guarantees that it will be computationally feasible to compute the primitive response partitions because only a small fraction of all possible parameter settings will need to be considered. If every parameter were relevant to a task, then every setting would have to be tried and this would not be feasible.

4. Region compactness: The regions of parameter space that must be considered must require only a small amount of memory space for their representation. ASPM represents sets of parameter settings in terms of cartesian products of parameter values. If a very large number of such cartesian products are required to represent a specific region, then that region may require an unacceptable amount of memory space. In the analyses we have done using ASPM this has never been a problem, but it could arise in the future.

Our belief is that many, and perhaps most, symbolic parameter models in cognitive science satisfy these requirements and that, as a result, ASPM should be an extremely useful and powerful set of computational tools.

References

Brown, J. S., & Burton, R. R. (1978). Diagnostic models for procedural bugs in basic mathematical skills. *Cognitive Science, 2*, 155-192.

Burton, R. R. (1982). Diagnosing bugs in a simple procedural skill. In Sleeman, D. H., & Brown, J. S. (Eds.), *Intelligent Tutoring Systems*. New York: Academic.

Polk, T.A. (1992). Verbal reasoning. Doctoral dissertation, Technical Report CMU-CS-92-178.

Polk, T.A. & Newell, A. (submitted). A verbal reasoning theory for categorical syllogisms. Unpublished.

VanLehn, K. (1990). *Mind bugs: The origins of procedural misconceptions*. Cambridge, MA: MIT Press.

VanLehn, K. and Pannu, A. (in preparation). Two techniques for reducing the combinatorial explosions of doing student modelling using ASPM. Unpublished manuscript.

Author Notes

This research was supported by the Office of Naval Research under contract number N00013-91-J-1527.

Notes

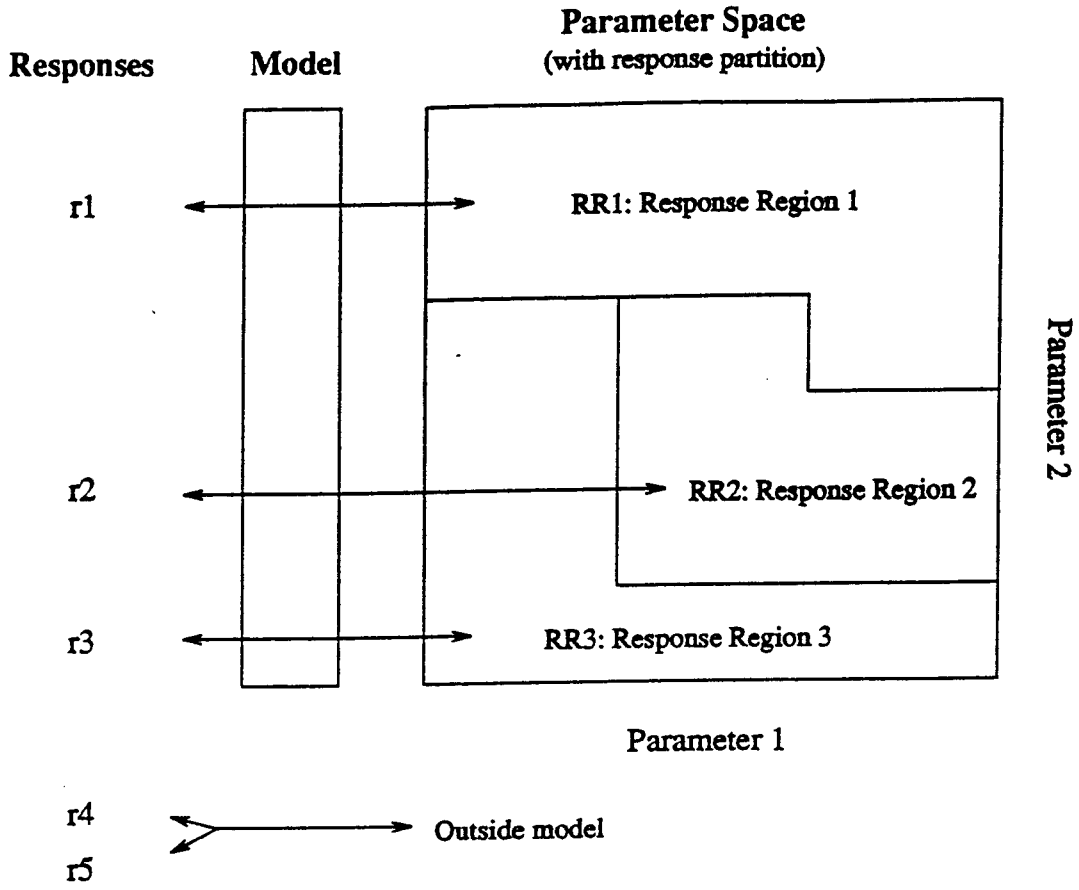
[1]A partition of a set is a collection of disjoint subsets whose union is equal to the set. We assume that a single response is associated with each parameter setting. Thus, a parameter setting can only be in one response region. Non-determinism can be handled by using sets of responses as the primitive responses.

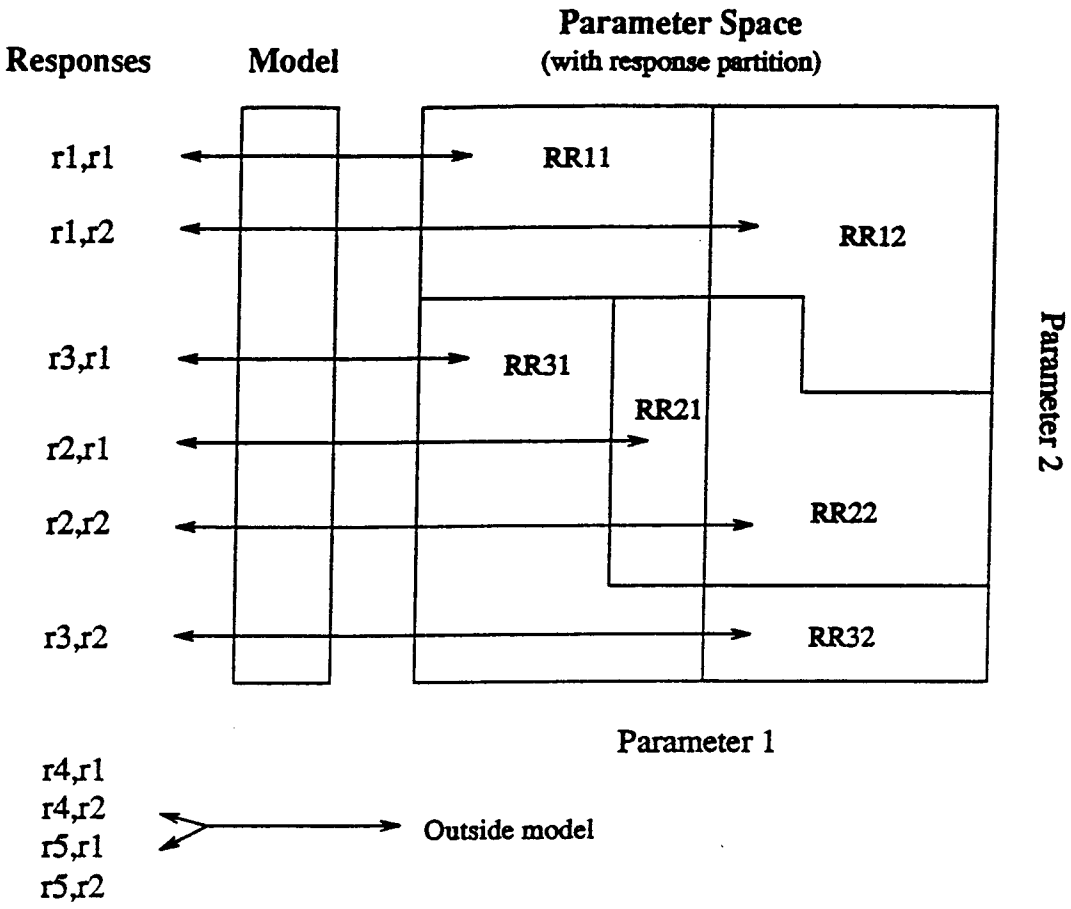
Figure Captions

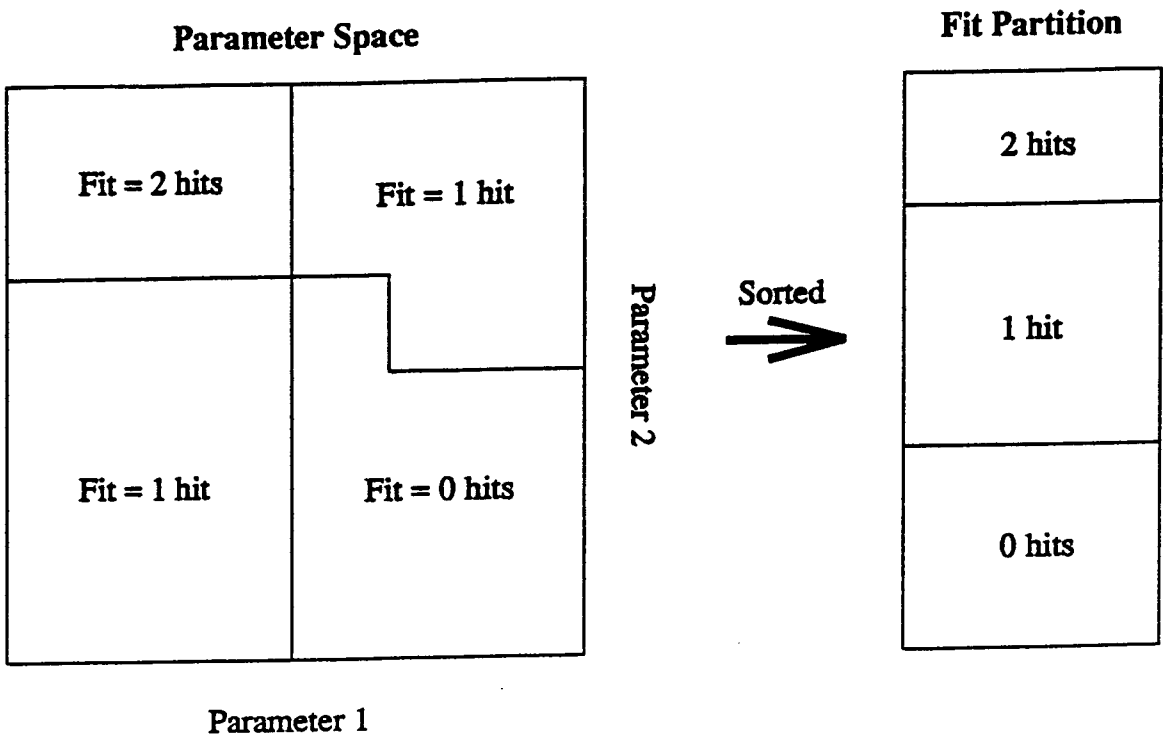
Figure 1. Parameter space, responses, a response partition, and response regions.

Figure 2. A compound response partition.

Figure 3. A fit partition and fit regions.


Parameter Space
(with response partition)
Responses
Model
r1
RR1: Response Region 1
r2
RR2: Response Region 2
r3
RR3: Response Region 3
Parameter 2
Parameter 1
r4
r5
Outside model


Parameter Space
(with response partition)
Responses
Model
r1,r1
r1,r2
r3,r1
r2,r1
r2,r2
r3,r2
RR11
RR12
RR31
RR21
RR22
RR32
Parameter 2
Parameter 1
r4,r1
r4,r2
r5,r1
r5,r2
Outside model


Parameter Space
Fit = 2 hits
Fit = 1 hit
Fit = 1 hit
Fit = 0 hits
Parameter 2
Parameter 1
Sorted
Fit Partition
2 hits
1 hit
0 hits